Spring 2024

1. a) Library Management System (14 marks)

=> তোমরা একটি Library Management System তৈরি করবে যেখানে Book, FictionBook, NonFictionBook এবং Library Class থাকবে।

=>

1.

```
class Book {                              [Book নামের Class]
    String title;                         [String variable থাকবে Title নামে]

public Book(String title) {               [Constructor]
    this.title = title;                   [Book এর নাম set করবে]
  }

public void read() {                      [Method, Book নাম Nameটা print করবে]
    System.out.println("Reading the book: " + title);
  }

public void displayDetails() {            [Method, Book Title জানাবে]
    System.out.println("Book Title: " + title);
  }
}
```

2. class FictionBook extends Book { [FictionBook class Book থেকে Inherit করছে]

String genre; [genre → বইয়ের ধরন সংরক্ষণ করবে]

public FictionBook(String title, String genre) { [Constructor]

super(title); → [এটা দিয়ে মূল Book class এর Constructor কে call করা হয়েছে]

this.genre = genre; → [এই লাইনটি FictionBook class এর genre (ধরন) variable এ user input genre এর মান সংরক্ষণ করে]

}

@override [methodটি Parent class এর একটি method কে override করছে]

```java
public void read() {
    System.out.println("Getting lost in the world of fiction.");
}
```

[যখন কেউ FictionBook class এর read() method call করবে, তখন এটি এই msg দেখাবে]

public void borrow() { [method, public তাই অন্য class থেকে accessable]

```java
    System.out.println("Borrowing the fiction book: " + title);
}
```

public void returnBook() { [method, বই ফেরত দেওয়াতে call করবে]

```java
    System.out.println("Returning the fiction book: " + title);
}
}
```

3.

```java
class NonFictionBook extends Book {
    String topic;                [topic variable-এ বইয়ের topic
                                  সংরক্ষণ করবে]

public NonFictionBook(String title, String topic) {
    super(title);                [Constructor]
    this.topic = topic;
}

@override [method টি Parent class-এর একটি method কে
           override করছে]
public void read() {
    System.out.println("Exploring the realm of non-
        fiction.");
}

public void borrow() {
    System.out.println("Borrowing the non-fiction
     book: " + title);
}

public void returnBook() {
    System.out.println("Returing the non-fiction book:"
     + title);
}
}
```

```java
public class Library {
public static void main (String[] args){
    Book myBook = new Book ("Java Programming");
    myBook.read();
```

[Library class যা মূল program চালাবে]

[myBook নামে একটি Book Object তৈরি করা হবে এবং read() method call করা হবে]

```java
FictionBook mysteryBook = new FictionBook("The Da Vinci Code", "Mystery");
 mysteryBook.read();
 mystery Book.borrow();
```

[mysteryBook নামে একটি FictionBook object তৈরি করা হবে, যা read() ও borrow() method call করবে]

```java
NonFictionBook historyBook = new NonFictionBook("Sapiens: A Brief History of Humankind", "History");
historyBook.read();
historyBook.returnBook();

  }
}
```

Reading the book: Java Programming
Getting lost in the world of fiction.
Borrowing the fiction book: The Da Vinci Code
Exploring the realm of non-fiction.
Returning the non-fiction book: Sapiens: A Brief History of Humankind.

CamScanner

1.b). Student class (Encapsulation Example) (6 marks)

আমাদের student class তৈরি করতে হবে যেখানে name, age এবং grade থাকবে। আমরা getter ও setter method ব্যবহার করে এগুলো access করব।

=>

```
class Student {                       [Student Class তৈরি করবে
                                      যা Encapsulation ব্যবহার
                                      করবে].
private String name;
private int age;
private double grade;

public void setName(String name){     [Name set করার
    this.name = name;                  method]
}

public String getname(){              [Name return করার
    return name;                       method]
}

public void setAge(int age){          [Age set করার
    this.age = age;                    method]
}
```

```
public int getAge(){        [Age return করার method]
    return age;
}

public void setGrade(double grade){    [Grade set করার method]
    this.grade = grade;
}

public double getGrade(){      [Grade return করার method]
    return grade;
  }
}
```

[Mainclass যেখানে আমরা Student class এর Object তৈরি করব]

```
public class Main {
  public static void main(String[] args){
    Student student = new Student();
    student.setName("Alice Wonderland");
    student.setAge(20);
    student.setGrade(95.5);

System.out.println("Name: " + student.getName());
System.out.println("Age:" + student.getAge());
System.out.println("Grade:" + student.getGrade());
}
}
```

CS CamScanner

Name: Alice Wonderland

Age: 20

Grade: 95.5

2.a) RoboHelper Virtual Assistant (Multiple Inheritance using Interface) (9 marks)

আমাদের একটি virtual Assistant (Robohelper) তৈরি করতে হবে, যা Teacher, Assistant এবং Student এর মতো আচরণ করে। [Interface এ শুধু method declare করা হয়, কোনো implementation থাকে না]

```
interface Teacher {                [Teacher interface তৈরি করলাম]
    void eating();                 [// Teacher eating method]
    void sleeping();               [// Teacher sleeping method]
}
```

[Assistant interface যা Teacher interface কে extend করবে]

```
interface Assistant extends Teacher {
    void assisting();              [Assistant হিসেবে সাহায্য করার method]
}
```

[Assistant interface Teacher কে extend করেছে, তাই এটি Teacher এর সকল method পেয়ে গেছে]

CamScanner

```
interface Student {            [Student interface তৈরি করা]
    void working();            [Student হিসেবে কাজ করার method]
}                              [Student interface এ working() method
                                রাখা হয়েছে]
```

[Human class Teacher, Assistant এবং Student Interface implement করবে]

```
class Human implements Assistant, Student {
    public void eating() {
    System.out.println("Teacher is eating.");
    }
```

[Assistant Interfaceটি Teacher Interface কে Extend করেছে, Human class Assistant interface implement করেছে, এর মানে Human class Assistant এর সাথে Teacher এর methodও পেয়ে গেছে]

```
    public void sleeping() {
    System.out.println("Teacher is sleeping.");
    }

    public void assisting() {
    System.out.println("Assistant is assisting.");
    }

    public void working() {
    System.out.println("Student is working.");
    }
}
```

CamScanner

[Human class তিনটি Interface Implement করেছে, তাই তাদের সব method এর ব্যাখ্যা দিতে বাধ্য]

[Main method]

```
public class RoboHelper {
    public static void main (String[] args) {
        Human helper = new Human ();

        helper.eating ();
        helper.sleeping ();
        helper.assisting ();
        helper.working ();
    }
}
```

[RoboHelper Class যা সব method call করবে]

[RoboHelper class এ আমরা Human এর Object তৈরি করে সব method call করেছি]

Teacher is eating.
Teacher is sleeping.
Assistant is assisting.
Student is working.

2.b) Exception Handling Code & Output Analysis (4 marks)

দেয়া codeটি ভুল আছে, তাই ঠিক code লিখে ব্যাখ্যা করলাম।

[JavaHungry class কে সবার জন্য accessable করে public করার মাধ্যমে]

```
public class JavaHungry {
  public static void main(String[] args) {
    int arr[] = {0, 1, 2, 3, 4, 5};
    try {
      int num = 99/0;
      System.out.print(arr[0]);
      System.out.print("A");
      int num2 = 99/0;
      System.out.print("B");
    }
    catch (ArithmeticException ex) {
      System.out.print("C");
    }
    catch (NumberFormatException ex) {
      System.out.print("D");
    }
```

[try block এর কাজ, যদি এমন code লিখি, যা run করার সময় Error (Exception) হতে পারে, যদি কোনো Exception হয়, তাহলে Catch Block execute হবে]

[এখানে, 99/0 করলে "Arithmetic Exception" (Divide by Zero) হবে]

[এই লাইনে error হবে, তাই পরের লাইনগুলো execute হবে না]

[এগুলো run হওয়ার কথা ছিলো, কিন্তু run হওয়ার আগের লাইনেই exception চলে আসায় run হবে না]

[এই Block এ শুধু তখনই আসবে, যখন ArithmeticException হয়, যেহেতু 99/0 ArithEx হয়েছে, তাই এখানে প্রবেশ করবে]

[C print করবে, কারণ exception হয়েছে]

[এটি NumberFormat সংক্রান্ত Exception ধরার জন্য, কিন্তু এই কোডে এই Exception হয় নি, তাই skip হয়ে যাবে]

```
catch (Exception ex) {
    System.out.print ("E");
}
finally {
    System.out.print("F");
}
System.out.print("G");
System.out.print (arr[4]);
}
}
```

[এটি Generic Exception ধরবে, যদি আগের গুলো না ধরে। যেহেতু Ari.Ex আগেই ধরেছে, তাই এটি skip হবে]

[Finally block সবসময় run হয়, Execute হয়, Exception হোক বা না হোক, F print হবে]

[try-catch block এর বাইরে যা আছে তা print হবে, G print হবে, arr[4] = Index 4 এর মান = 4, তাই arr[4] = 4 print হবে]

Output

C F G 4

Step by step		
ধাপ	Execution	Output
1	99/0; → ArithmeticException	
2	Exception হওয়ার কারণে catch (Ari.Ex ex) এ যাবে	
3.	"C" print করবে	C
4.	finally Block এ যাবে	
5.	"F" print করবে	C F
6.	Try-catch Block এর বাইরে "G" print করবে	C F G
7.	arr[4] print করবে (যার মান 4)	CFG4

CamScanner

2.c) Error Types with Examples (3 marks)

1. Syntax Error (Compilation Error)

=> যখন code এ ভুল থাকে এবং এটি compile হয় না।

Example:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello World" //*Missing closing parenthesis**
    }
}
```

Error: ';' expected

2. Runtime Error (Exception)

=> যখন program runtime এ crash হয়।

Example:

```
public class Test {
    public static void main(String[] args) {
        int num = 5/0 ; //** Division by Zero, ArithmeticException হবে। ***
    }
}
```

3. Logical Error

⇒ যখন program run হয় কিন্তু wrong output দেয়।

Example:

```
public class Test {
    public static void main(String[] args) {
        int sum = 5 * 10; // **গুণ করা হয়েছে, দরকার ছিলো যোগ করা**
        System.out.println("Sum: " + sum); // wrong output
    }
}
```

সঠিক কোড: int sum = 5+10;

2. d) Custom Exception (Invalid Age Exception):

1. Custom Exception তৈরি করা

[আমরা Exception class থেকে Inherit (বংশগতভাবে গ্রহণ) করে Custom Exception তৈরি করবো]

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}
```

↳ [Constructor, যা একটি error message গ্রহণ করবে]

↳ [Exception class এর মধ্যে message পাঠাচ্ছি, যাতে এটি getMessage() method দিয়ে error message দেখাতে পারে]

[checkAge() method একটি age গ্রহণ করবে এবং InvalidAgeException ছুঁড়ে দিতে পারে বলে ঘোষণা করা হয়েছে]

2. LibrarySystem class এবং checkAge() method

```java
class LibrarySystem {
    public static void checkAge(int age) throws
    InvalidAgeException {
        if (age < 12) {
            throw new InvalidAgeException("Error: Age must
be at least 12 years old to borrow books.");
        } else {
            System.out.println("Age is valid for borrowing.");
        }
    }
}
```

[Age 12 এর কম হলে নতুন InvalidAgeException তৈরি করা হয়েছে এবং error message পাঠানো হয়েছে]

[বেশি হলে এটা print করবে]

3. main() method এবং Exception Handling

```java
public static void main(String[] args) {
    try {
        checkAge(10);  //** Invalid Age Exception হবে **
    } catch (InvalidAgeException e) {
        System.out.println(e.getMessage());
    }
}
```

[যখন Exception ধরা পড়বে, তখন e.getMessage() দিয়ে Error message print হবে]

[10 বছর দিয়ে checkAge() call করা হয়েছে (যা InvalidAgeException Trigger করবে]

Output: Error: Age must be at least 12 years old to borrow books.

কারণ, checkAge(10) call করা হয়েছে এবং ১০ বছর ১২ বছরের কম, তাই InvalidAgeException ছুঁড়ে দেওয়া হয়েছে এবং error message print হয়েছে।

<u>Summary</u>

1. Custom Exception তৈরি করা হয়েছে।
2. checkAge( ) method বয়স check করে Exception ছুঁড়ে দিতে পারে।
3. main() method এ try-catch ব্যবহার করা হয়েছে, যাতে InvalidAgeException ধরা পড়ে।
4. যেহেতু ১০ < ১২, তাই Exception ছোঁড়া হয়েছে এবং error message print হয়েছে।